Introduction to SQL

Class 16

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, REPEAT, LOOP), ,Stored functions ,Stored Procedures
  - Views
  - Indexes, Cursors, Triggers, Events
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases
  - ACID, CAP
  - How SQL databases internally works

TRANSACTIONS

- A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit
- In other words, a transaction will never be complete unless each individual operation within the group is successful
- If any operation within the transaction fails, the entire transaction will fail.

Real life examples of transactions

- Banking operations: Transactions can be used to ensure that banking operations, such as transferring funds from one account to another, are executed atomically and in a consistent manner.
- Online shopping: Transactions can be used to ensure that a customer's order is properly recorded in the database, including updating the inventory and creating an order record. This way, even if there is a failure in the middle of the process, the transaction can be rolled back to avoid partial updates or inconsistencies in the database.
- Stock market trading: Transactions can be used to ensure that a stock trade is executed atomically and in a consistent manner. For example, if a trade involves buying and selling stocks, the transaction ensures that either both operations succeed or both fail, avoiding partial updates or inconsistencies in the database.
- Flight booking: Transactions can be used to ensure that a flight booking is properly recorded in the database, including updating the available seats and creating a booking record. This way, even if there is a failure in the middle of the process, the transaction can be rolled back to avoid partial updates or inconsistencies in the database.
- Health care management: Transactions can be used to ensure that patient information is recorded atomically and in a consistent manner, avoiding partial updates or inconsistencies in the database.

Properties of transactions

- Transactions have the following four standard properties, usually referred to by the acronym **ACID**
- **Atomicity** – This ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.
- **Consistency** – This ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation** – This enables transactions to operate independently on and transparent to each other.
- **Durability** – This ensures that the result or effect of a committed transaction persists in case of a system failure

Atomicity

- This property ensures that all statements or operations within the transaction unit must be executed successfully. Otherwise, if any operation is failed, the whole transaction will be aborted, and it goes rolled back into their previous state. It includes features:
  - COMMIT statement.
  - ROLLBACK statement.
  - Auto-commit setting.
  - Operational data from the INFORMATION_SCHEMA tables.

Consistency

- This property ensures that the database changes state only when a transaction will be committed successfully. It is also responsible for protecting data from crashes. It includes features:
  - InnoDB doublewrite buffer.
  - InnoDB crash recovery.

Isolation

- This property guarantees that each operation in the transaction unit operated independently. It also ensures that statements are transparent to each other. It includes features:
  - SET ISOLATION LEVEL statement.
  - Auto-commit setting.
  - The low-level details of InnoDB locking.

Durability

- This property guarantees that the result of committed transactions persists permanently even if the system crashes or failed. It includes features:
  - Write buffer in a storage device.
  - Battery-backed cache in a storage device.
  - Configuration option innodb_file_per_table.
  - Configuration option innodb_flush_log_at_trx_commit.
  - Configuration option sync_binlog.

COMMIT,ROLLBACK,SAVEPOINT

- In MySQL, the transactions begin with the statement **BEGIN WORK** and end with either a **COMMIT** or a **ROLLBACK** statement
- The SQL commands between the beginning and ending statements form the bulk of the transaction
- **COMMIT** - When a successful transaction is completed, the COMMIT command should be issued so that the changes to all involved tables will take effect.
- **ROLLBACK** - If a failure occurs, a ROLLBACK command should be issued to return every table referenced in the transaction to its previous state.

AUTOCOMMIT

- You can control the behavior of a transaction by setting session variable called AUTOCOMMIT
- If AUTOCOMMIT is set to 1 (the default), then each SQL statement (within a transaction or not) is considered a complete transaction and committed by default when it finishes
- When AUTOCOMMIT is set to 0, by issuing the **SET AUTOCOMMIT = 0** command, the subsequent series of statements acts like a transaction and no activities are committed until an explicit COMMIT statement is issued.

A Generic Example on Transaction

- Begin transaction by issuing the SQL command BEGIN WORK.
- Issue one or more SQL commands like SELECT, INSERT, UPDATE or DELETE.
- Check if there is no error and everything is according to your requirement.
- If there is any error, then issue a ROLLBACK command, otherwise issue a COMMIT command.

COMMIT Example

```
-- 1. Start a new transaction
START TRANSACTION;
-- 2. Get the highest income
SELECT @income:= MAX(income) FROM employees;
-- 3. Insert a new record into the employee table
INSERT INTO employees(emp_id, emp_name, emp_age, city, income)
VALUES (111, 'Alexander', 45, 'California', 70000);
-- 4. Insert a new record into the order table
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');
-- 5. Commit changes
COMMIT;
```

ROLLBACK Example

```
-- 1. Start a new transaction
START TRANSACTION;
-- 2. Delete data from the order table
DELETE FROM Orders;
-- 3. Rollback changes
ROLLBACK;
-- 4. Verify the records in the first session
SELECT * FROM Orders;
```

Statements that cannot be a rollback in using MySQL Transaction.

- MySQL Transaction cannot be able to roll back all statements. For example, these statements include DDL (Data Definition Language) commands such as CREATE, ALTER, or DROP database as well as CREATE, UPDATE, or DROP tables or stored routines. We have to make sure that when we design our transaction, these statements do not include.

SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT

- The SAVEPOINT statement creates a special mark with the name of the identifier inside a transaction.
- It allows all statements that are executed after savepoint to be rolled back. So that the transaction restores to the previous state it was in at the point of the savepoint
- The ROLLBACK TO SAVEPOINT statement allows us to rolls back all transactions to the given savepoint was established without aborting the transaction.
- The RELEASE SAVEPOINT statement destroys the named savepoint from the current transaction without undoing the effects of queries executed after the savepoint was established.

SAVEPOINT savepoint_name

ROLLBACK TO [SAVEPOINT] savepoint_name

RELEASE SAVEPOINT savepoint_name

ROLLBACK TO SAVEPOINT example

```
START TRANSACTION;
SELECT * FROM Orders;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');
SAVEPOINT my_savepoint;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (7, 'Ink', 5894, '2020-03-10');
ROLLBACK TO SAVEPOINT my_savepoint;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (8, 'Speaker', 6065, '2020-02-18');
COMMIT;
```

RELEASE SAVEPOINT

```
START TRANSACTION;
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (7, 'Ink', 5894, '2020-03-10');
SAVEPOINT my_savepoint;
UPDATE Orders SET prod_name='Scanner' WHERE order_id=8;
RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

Handling errors in mysql

- DECLARE action HANDLER FOR condition_value statement;

If a condition whose value matches the `condition_value` , MySQL will execute the `statement` and continue or exit the current code block based on the `action` .

The `action` accepts one of the following values:

- `CONTINUE` : the execution of the enclosing code block (`BEGIN` ... `END`) continues.
- `EXIT` : the execution of the enclosing code block, where the handler is declared, terminates.

The `condition_value` specifies a particular condition or a class of conditions that activate the handler. The `condition_value` accepts one of the following values:

- A MySQL error code.
- A standard `SQLSTATE` value. Or it can be an `SQLWARNING` , `NOTFOUND` or `SQLEXCEPTION` condition, which is shorthand for the class of `SQLSTATE` values. The `NOTFOUND` condition is used for a cursor or `SELECT INTO variable_list` statement.
- A named condition associated with either a MySQL error code or `SQLSTATE` value.

The `statement` could be a simple statement or a compound statement enclosing by the `BEGIN` and `END` keywords.

Examples

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
SET hasError = 1;
```

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'An error has occurred, operation rollbacked and the stored procedure was terminated';
END;
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET RowNotFound = 1;
```

```
DECLARE CONTINUE HANDLER FOR 1062
SELECT 'Error, duplicate key occurred';
```

```
CREATE TABLE SupplierProducts (
    supplierId INT,
    productId INT,
    PRIMARY KEY (supplierId , productId)
);
```

```
CREATE PROCEDURE InsertSupplierProduct(
   IN inSupplierId INT,
   IN inProductId INT
)
BEGIN
   -- exit if the duplicate key occurs
   DECLARE EXIT HANDLER FOR 1062
   BEGIN
                SELECT CONCAT('Duplicate key (',inSupplierId,',',inProductId,') occurred') AS message;
   END;
   -- insert a new row into the SupplierProducts
   INSERT INTO SupplierProducts(supplierId,productId)
   VALUES(inSupplierId,inProductId);
   -- return the products supplied by the supplier id
   SELECT COUNT(*)
   FROM SupplierProducts
   WHERE supplierId = inSupplierId;
END$$
DELIMITER ;
```

```
DROP PROCEDURE IF EXISTS InsertSupplierProduct;
DELIMITER $$
CREATE PROCEDURE InsertSupplierProduct(
   IN inSupplierId INT,
   IN inProductId INT
)
BEGIN
   -- exit if the duplicate key occurs
   DECLARE CONTINUE HANDLER FOR 1062
   BEGIN
                SELECT CONCAT('Duplicate key (',inSupplierId,',',inProductId,') occurred') AS message;
   END;
   -- insert a new row into the SupplierProducts
   INSERT INTO SupplierProducts(supplierId,productId)
   VALUES(inSupplierId,inProductId);
   -- return the products supplied by the supplier id
   SELECT COUNT(*)
   FROM SupplierProducts
   WHERE supplierId = inSupplierId;
END$$
DELIMITER ;
```

MySQL handler precedence

In case you have multiple handlers that handle the same error, MySQL will call the most specific handler to handle the error first based on the following rules:

- An error always maps to a MySQL error code because in MySQL it is the most specific.
- An `SQLSTATE` may map to many MySQL error codes, therefore, it is less specific.
- An `SQLEXCPETION` or an `SQLWARNING` is the shorthand for a class of `SQLSTATES` values so it is the most generic.

Based on the handler precedence rules, MySQL error code handler, `SQLSTATE` handler and `SQLEXCEPTION` takes the first, second and third precedence.

Suppose that we have three handlers in the handlers in the stored procedure `insert_article_tags_3` :

Using a named error condition

```
DELIMITER $$

CREATE PROCEDURE TestProc()
BEGIN

    DECLARE EXIT HANDLER FOR 1146
    SELECT 'Please create table abc first' Message;

    SELECT * FROM abc;
END$$

DELIMITER ;
```

```
DROP PROCEDURE IF EXISTS TestProc;

DELIMITER $$

CREATE PROCEDURE TestProc()
BEGIN
   DECLARE TableNotFound CONDITION for 1146 ;

   DECLARE EXIT HANDLER FOR TableNotFound
         SELECT 'Please create table abc first' Message;
   SELECT * FROM abc;
END$$

DELIMITER ;
```

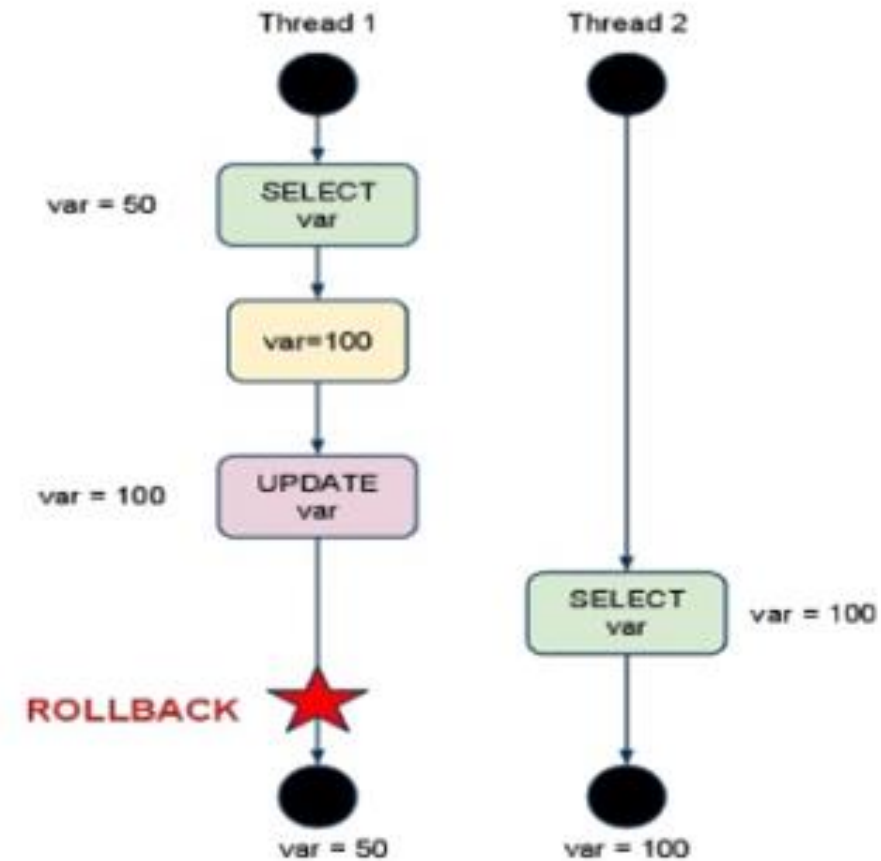
Isolation Levels for a transaction In MySQL

- There are 4 levels of isolation supported by mysql
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE
- The Isolation level’s can be set globally or session based on our requirements.

Read Uncommitted

- In READ-UNCOMMITTED isolation level, there isn’t much isolation present between the transactions at all, ie ., No locks
- A transaction can see changes to data made by other transactions that are not committed yet
- A transaction can see changes to data made by other transactions that are not committed yet
- With this isolation level, there is always for getting a “Dirty-Read”

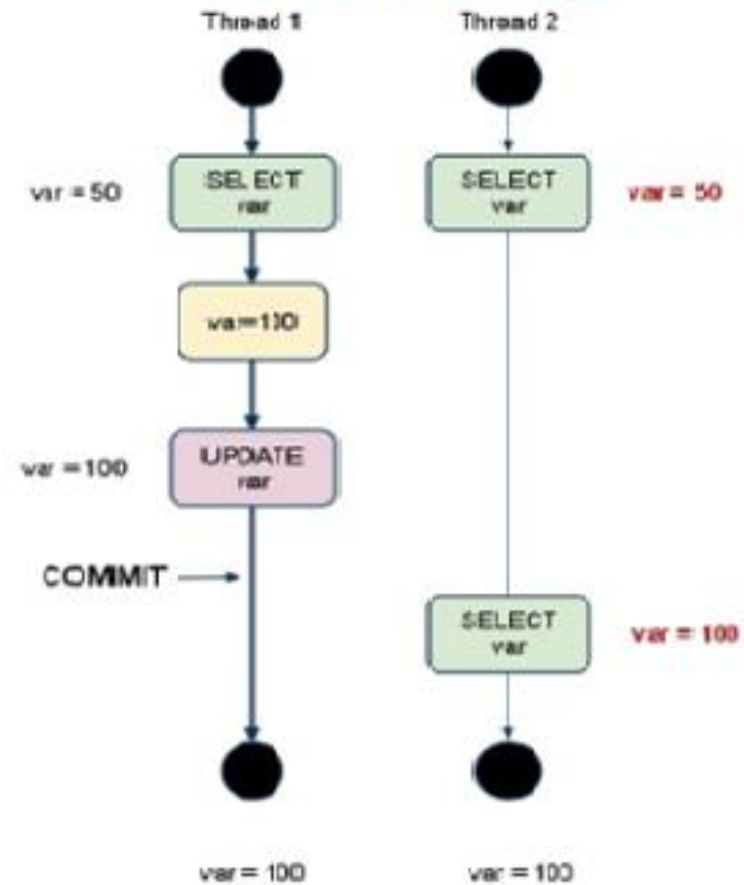
DIRTY READS


Thread 1
Thread 2
var = 50
SELECT var
var=100
var = 100
UPDATE var
SELECT var
var = 100
ROLLBACK
var = 50
var = 100

READ COMMITTED

- IN READ-COMMITTED isolation level, the phenomenon of dirty read is avoided, because any uncommitted changes are not visible to any other transaction until the change is committed
- This is the default isolation level with most of popular RDBMS software, but not with MySQL.
- Within this isolation level, each SELECT uses its own snapshot of the committed data that was committed before the execution of the SELECT
- Now because each SELECT has its own snapshot, here is the trade-off now, so the same SELECT, when running multiple times during the same transaction, could return different result sets. This phenomenon is called non-repeatable read.

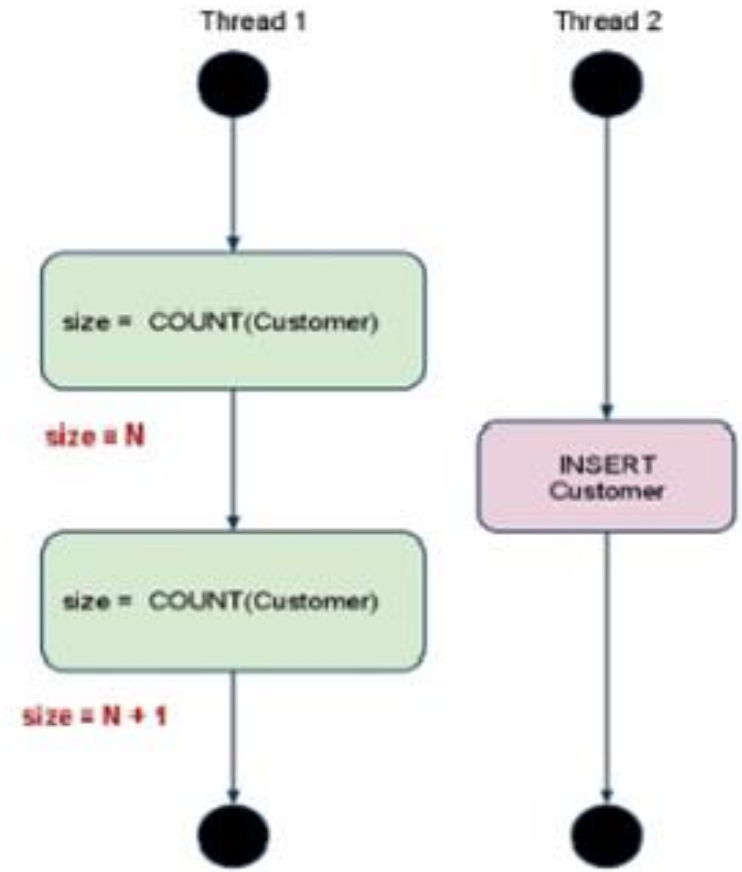
NON REPEATABLE READS



REPEATABLE READ

- In REPEATABLE-READ isolation level, the phenomenon of non-repeatable read is avoided.
- It is the default isolation in MySQL.
- This isolation level returns the same result set throughout the transaction execution for the same SELECT run any number of times during the progression of a transaction.
- A snapshot of the SELECT is taken the first time the SELECT is run during the transaction and the same snapshot is used throughout the transaction when the same SELECT is executed.
- A transaction running in this isolation level does not take into account any changes to data made by other transactions, regardless of whether the changes have been committed or not.
- This ensures that reads are always consistent(repeatable)
- Maintaining a snapshot can cause extra overhead and impact some performance
- Although this isolation level solves the problem of non-repeatable read, another possible problem that occurs is phantom reads.

PHANTOM READS



SERIALIZABLE

- SERIALIZABLE completely isolates the effect of one transaction from others
- It is similar to REPEATABLE READ with the additional restriction that row selected by one transaction cannot be changed by another until the first transaction finishes
- The phenomenon of phantom reads is avoided
- This isolation level is the strongest possible isolation level

What are MySQL Locks?

- A MySQL Locks is nothing but a flag that can be assigned to a table to alter its properties. MySQL allows a table lock that can be assigned by a client-server to prevent other sessions from being able to access the same table during a specific time frame.

LOCK TABLES

table_name [[AS] alias] lock_type

[, tbl_name [[AS] alias] lock_type] ...

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

lock a table using an alias, you must refer to it in your statements using that alias

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES

mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

What are 2 Types of MySQL Locks?

- MySQL Locks: Read Locks
- MySQL Locks: Write Locks

Features of the READ Lock

- MySQL allows multiple sessions to bi in READ lock for a table concurrently. And it also allows for other sessions to read the table without acquiring the lock.
- If the session holds the READ lock on a table, they cannot perform a write operation on it. It is because the READ lock can only read data from the table. All other sessions that do not acquire a READ lock are not able to write data into the table without releasing the READ lock. The write operations go into the waiting states until we have not released the READ lock.
- When the session is terminated normally or abnormally, MySQL implicitly releases all types of locks onto the table. This feature is also relevant for the WRITE lock

- WRITE locks have higher priority when compared to READ locks to ensure that updates are processed asap.
- This means that if a session has obtained a READ lock and simultaneously another session requests a WRITE lock, the session with READ lock requests has to wait until the session that requested the WRITE the lock has obtained the lock and released it.

Features of WRITE locks

- It is the session that holds the lock of a table and can read and write data both from the table.
- It is the only session that accesses the table by holding a lock. And all other sessions cannot access the data of the table until the WRITE lock is released.

How to unlock the table in MySQL?

- If a session issues a LOCK TABLES statement to acquire a lock while already holding MySQL locks, its existing locks are released implicitly before the new locks get granted.
- A session can release its locks explicitly with the help of UNLOCK TABLES.
- If a session begins a transaction, an implicit UNLOCK TABLES is performed, which causes existing MySQL locks to be released.
- If you lock a table explicitly with LOCK TABLES, any tables used in triggers also get locked implicitly:
- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.

Data modelling and Database design

What is a data model?

- A data model is a simplified representation of a real-world system or phenomenon that is used to describe and understand the data
- It is a way of organizing and structuring data to effectively and efficiently store, process, and retrieve information.
- Data models can take many forms, such as a relational database model, a network model, an object-oriented model, and more.

Objectives

- Identify and describe important entities and relationships to model data
- Develop data models to represent, organize, and store data
- Design and use relational databases to organize, store, and manipulate data

Data Modeling

- Goal – make sure all data objects required by a database are completely and accurately represented
- Data model design – the blueprint for creating a physical implementation of a database

Data Modeling Terms

- **Entity** – a class of real world objects having common attributes (e.g., sites, variables, methods).
- **Attribute** – A characteristic or property of an entity (site name, latitude, longitude)
- **Relationship** – an association between two or more entities
- **Cardinality** – the number of entities on either end of a relationship (one-to-one, one-to-many, many-to-many, etc.)

Data Modeling Exercise

- Consider:
  - What is the “entity”?
  - What are the “attributes” the entity?

Data Modeling Exercise

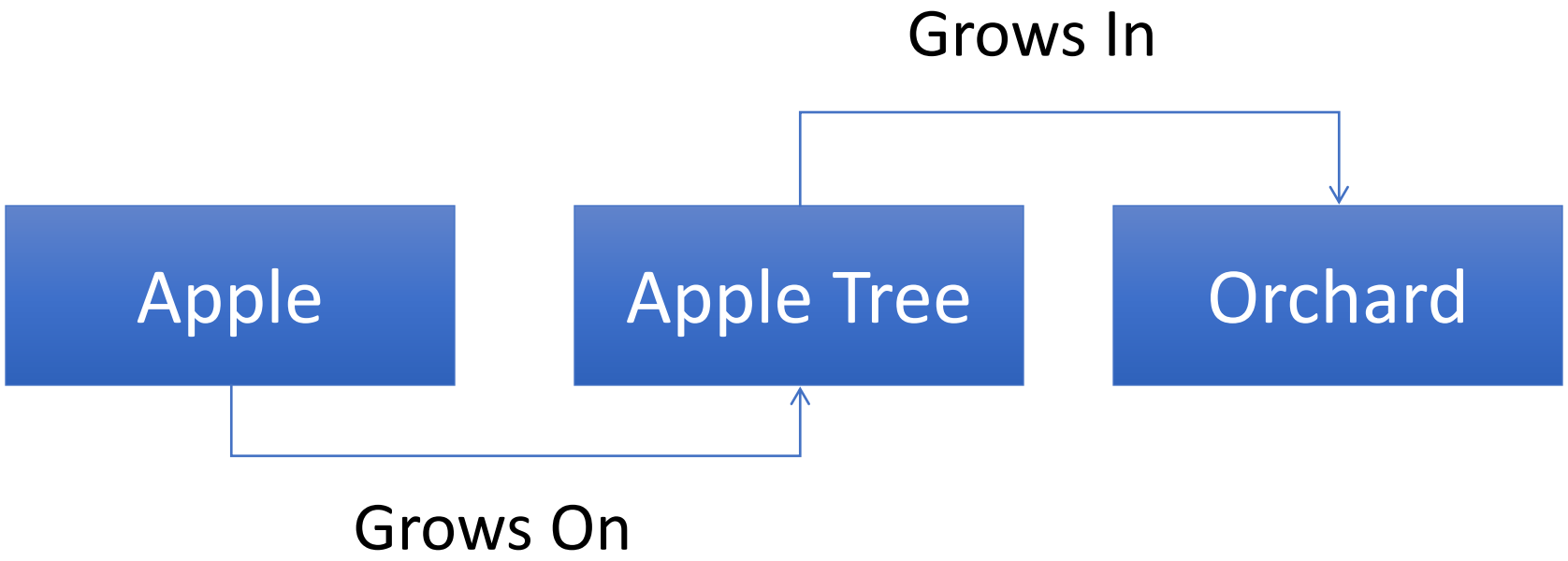
- What is the entity?
- What are the attributes?

Data Modeling Exercise

- What is the entity?
- What are the attributes?

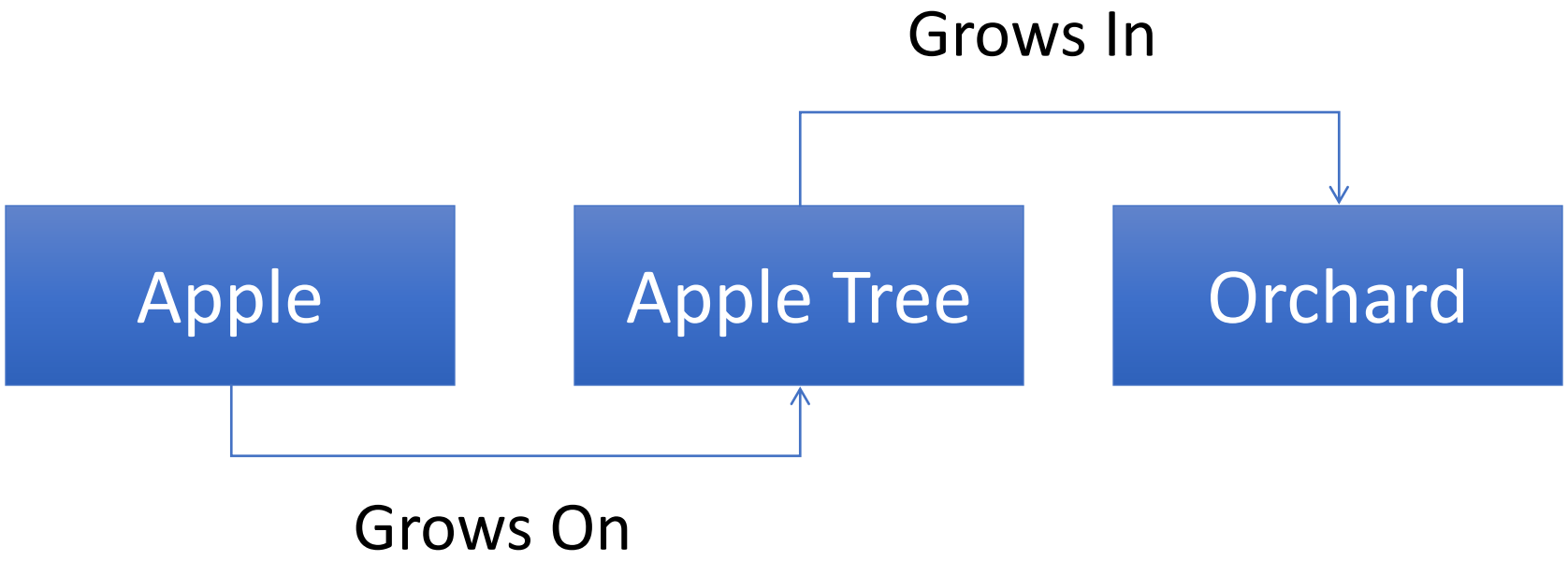
Data Modeling Exercise

- What are the relationships?



Data Modeling Exercise

- What are the relationships?



What about the business rules?

Data Model Requirements

- What is the information/data domain that you are modeling?
- What are the 20 queries that you want to do?
  - e.g., "Give me simultaneous observations of turbidity and TSS collected during the spring snowmelt period so I can develop a regression in R."
- What software do you want (have) to use?
- How do you want to share the data?

Data Model Design

- Our focus – relational data model design
- Three stages:
  - Conceptual data model
  - Logical data model
  - Physical data model

Conceptual Data Model (AKA – The Information Model)

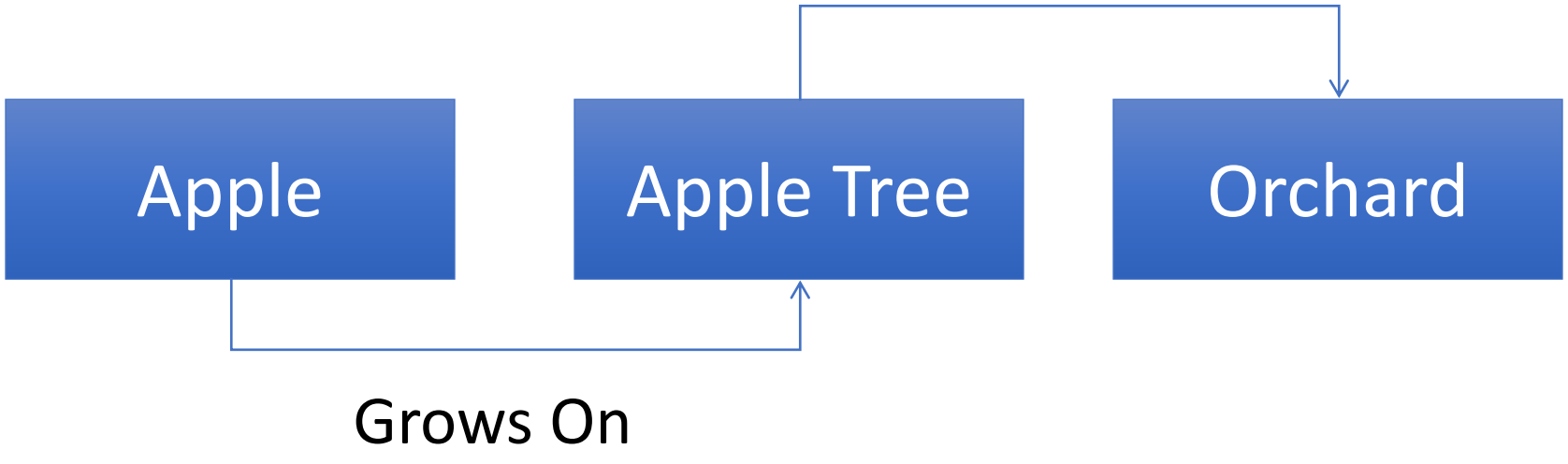
- High-level description of the data domain
- Does not constrain how that description is mapped to an actual implementation in software
- There may be many mappings
  - Relational database
  - Object model
  - XML schema, etc.

Goals of Conceptual Data Modeling

- The goals of conceptual data modeling are:
  - To capture the essential and intrinsic characteristics of data, independent of the physical storage or technology used to manage the data.
  - To create a common understanding and representation of the data among stakeholders, such as business analysts, data analysts, and IT personnel.
  - To identify and define the relationships and dependencies between data entities.
  - To support the design of effective and efficient data storage, retrieval, and processing systems.
  - To support the evolution and growth of the data and information systems over time.
- Overall, conceptual data modeling is a critical step in the data management process that helps organizations understand and manage their data more effectively.

Apple/Tree/Orchard Conceptual Model

Grows In



Logical Data Model

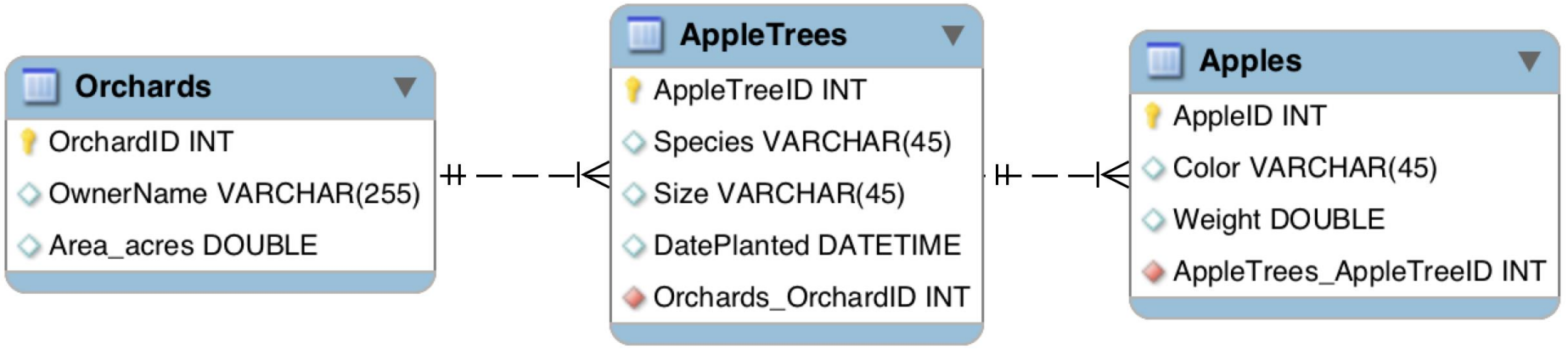
- Technology independent
- Contains more detail than the Conceptual Data Model
- Considered by many to be just an expanded conceptual data model
- Defines
  - Entities AND their attributes
  - Relationships AND cardinality
  - Constraints
- Generally completed as a documented Entity Relationship (ER) Model or diagram

Goals of Logical Data Modeling

- The goals of logical data modeling are:
  - To refine the conceptual data model and define a more detailed representation of the data, including specific data elements, data types, and relationships between entities.
  - To ensure that the data model accurately reflects the business requirements and meets the needs of the stakeholders.
  - To provide a consistent and unambiguous representation of the data that can be used as a blueprint for physical database design and implementation.
  - To provide a clear definition of the business rules and constraints that apply to the data.
  - To facilitate the integration of data from multiple sources into a unified data model.
- Overall, logical data modeling is a critical step in the data management process that helps organizations design and implement effective and efficient data management systems.

Entity Relationship Diagram

- Documentation of the structure of the data
- Used to communicate the design
- Serve as the basis for data model implementation



Advantages of having Entity Relationship Diagram

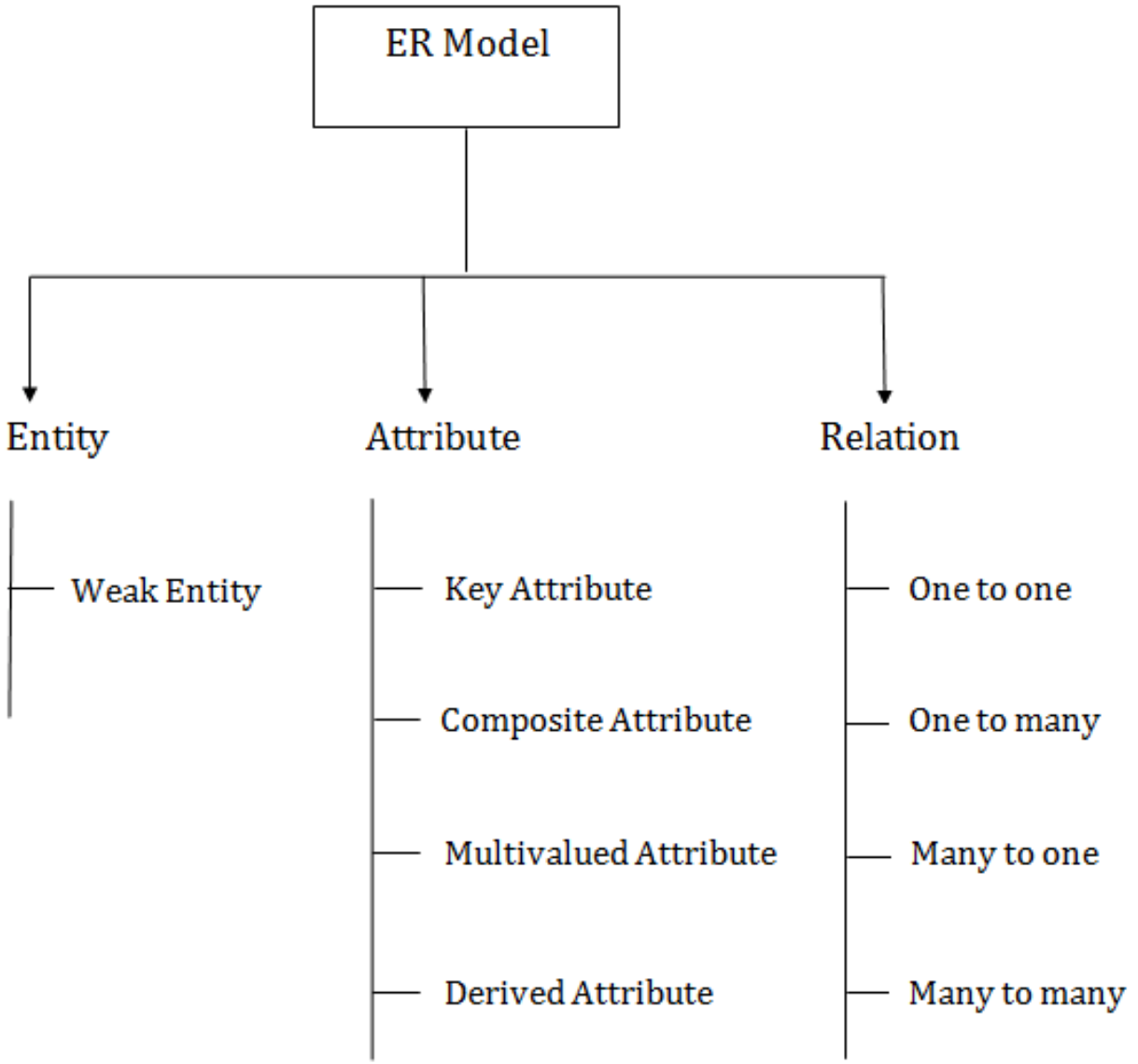
- The Entity Relationship (ER) diagram has several advantages, including:
  - Improved Communication: ER diagrams provide a visual representation of the data model that can be easily understood by stakeholders, including business analysts, data analysts, and IT personnel. This improved communication helps to ensure that the data model accurately reflects the business requirements.
  - Better Data Modeling: ER diagrams provide a systematic and structured approach to data modeling that helps to identify entities, relationships, and constraints. This leads to a more accurate and complete representation of the data.
  - Enhanced Data Integrity: ER diagrams define the relationships between entities, which helps to ensure that the data is stored and processed in a consistent and correct manner. This leads to enhanced data integrity and reduces the risk of data inconsistencies and errors.
  - Better Database Design: ER diagrams provide a basis for physical database design and implementation. They help to ensure that the database design is optimized for performance and scalability, and that it meets the needs of the stakeholders.
  - Improved Data Management: ER diagrams provide a clear and concise representation of the data that can be used as a reference for data management activities, such as data warehousing, data integration, and data quality management.
- Overall, ER diagrams provide a powerful tool for data modeling and management that can help organizations to better understand and manage their data.

Entity Relationship Diagram
(Relation Database Context)

- Entities effectively become tables
- Attributes describe entities and become fields (columns) in tables
- Relationships link tables on a common attribute or “key” and become formal constraints (part of the business rules)

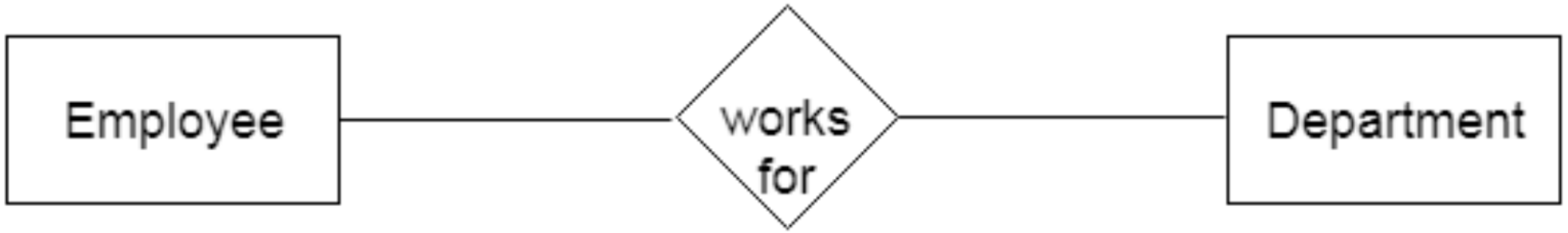
Main Elements of an Entity-Relationship Diagram

- Entities: Entities are objects or concepts that exist in the real-world and have a unique identity. They are represented as boxes or rectangles in an ER diagram.
- Attributes: Attributes are characteristics or properties of entities. They are represented as ellipses within the entity rectangle in an ER diagram.
- Relationships: Relationships describe the association between two or more entities. They are represented as a diamond shape connecting two or more entities in an ER diagram.
- Cardinality: Cardinality defines the number of instances of one entity that are associated with one instance of another entity. It is represented as an arrowhead on the relationship line in an ER diagram.
- Optionality: Optionality defines whether the relationship between two entities is mandatory or optional. It is represented by a circle or an open or filled diamond at the end of the relationship line in an ER diagram.
- Key Constraints: Key constraints define the unique identifier for each entity. They are represented by underlining the attribute name in an ER diagram.


ER Model
Entity
Attribute
Relation
Weak Entity
Key Attribute
Composite Attribute
Multivalued Attribute
Derived Attribute
One to one
One to many
Many to one
Many to many

ER Diagram Entity Notation

ENTITY



Regular or Strong & Weak entities (Independent and dependent entities)

- Strong or Regular Entities: Strong entities have a unique identifier called a primary key and can exist independently in the database. They do not rely on other entities to identify or describe them.
- Weak Entities: Weak entities do not have a primary key and cannot exist independently in the database. They are dependent on another entity, called the owner entity, to identify or describe them. The combination of the primary key of the owner entity and the weak entity's partial key forms a unique identifier for the weak entity.

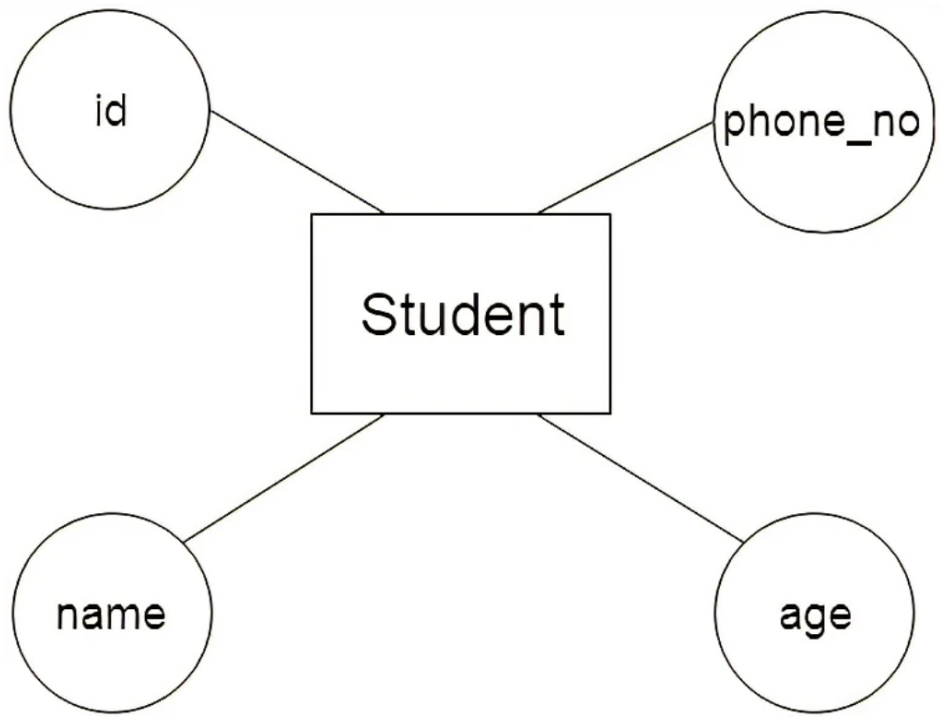
a. Weak Entity

An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.

2. Attribute

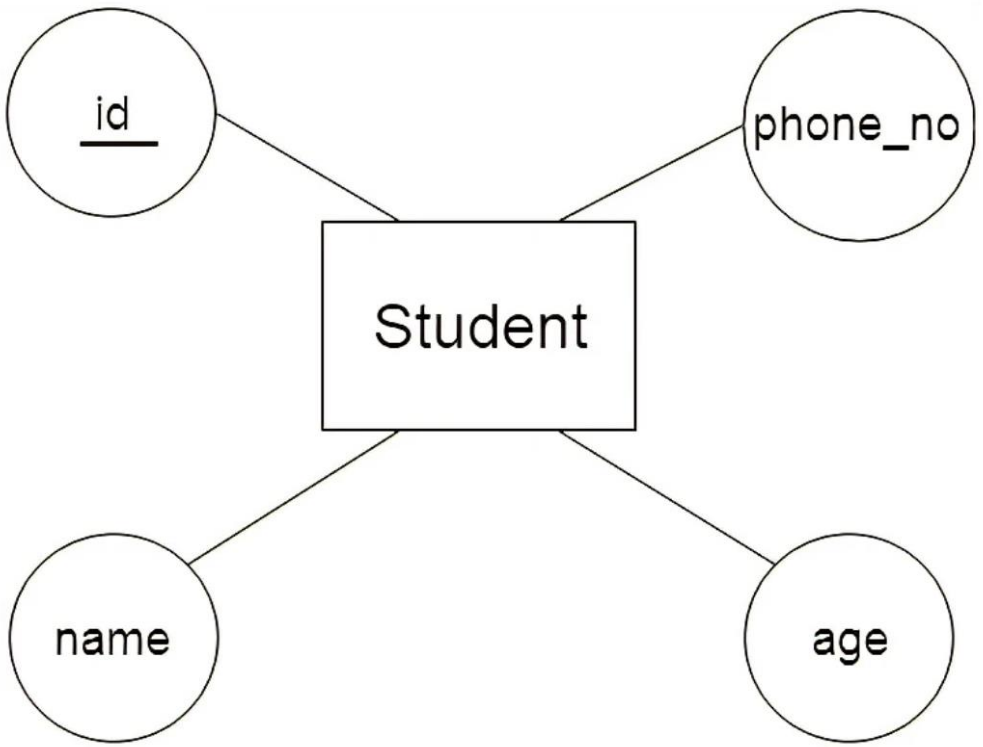
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



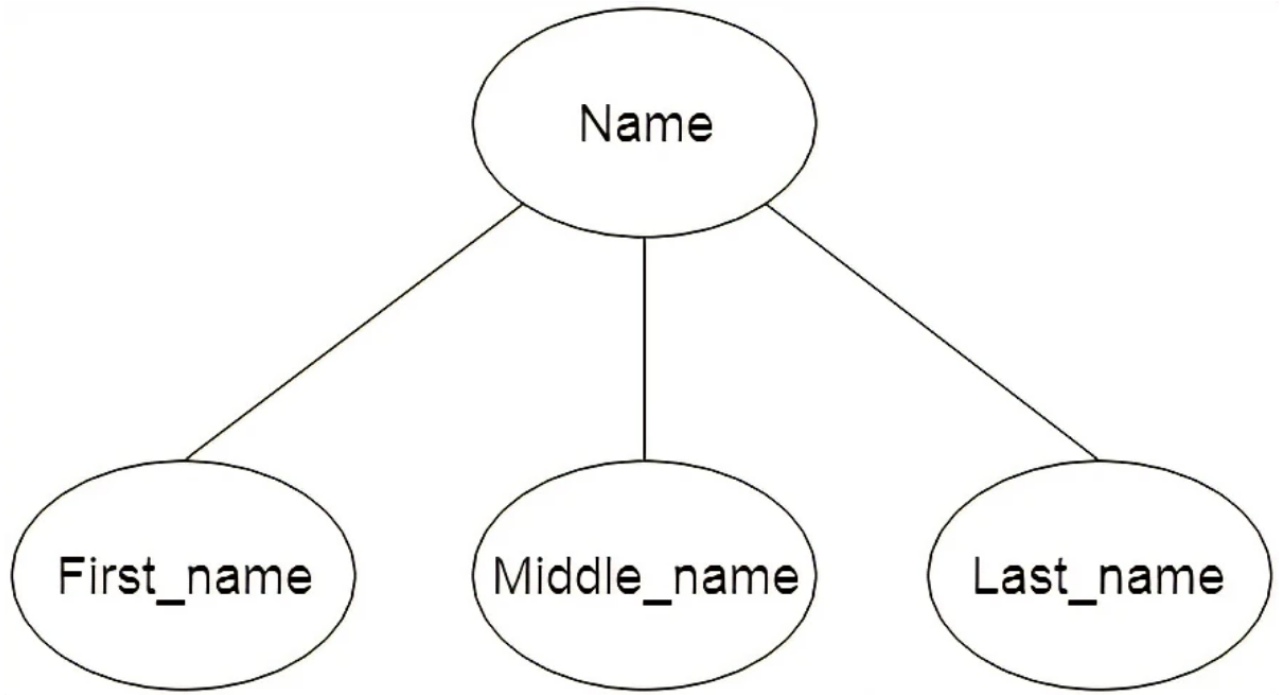
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

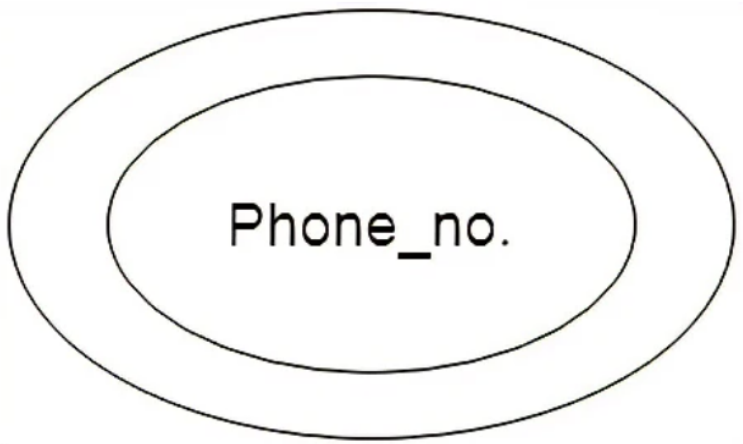
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

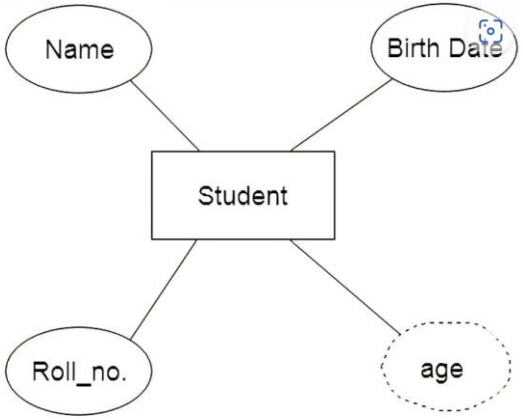
For example, a student can have more than one phone number.



d. Derived Attribute

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

a. One-to-One Relationship

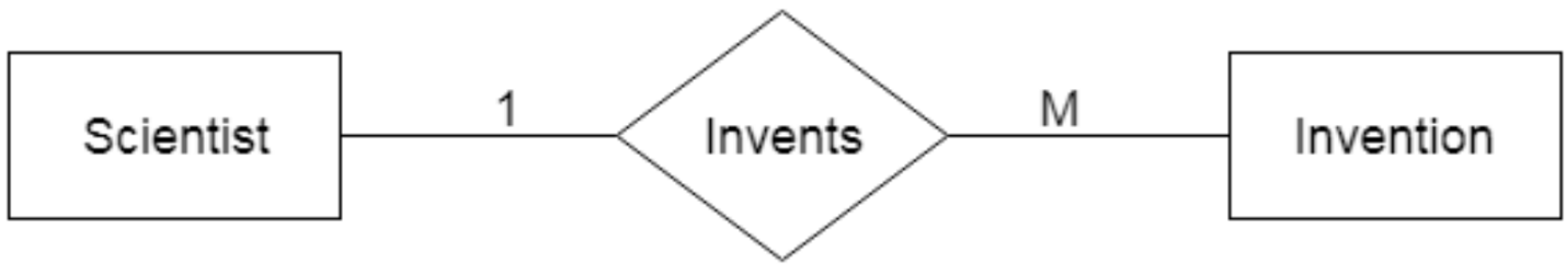
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

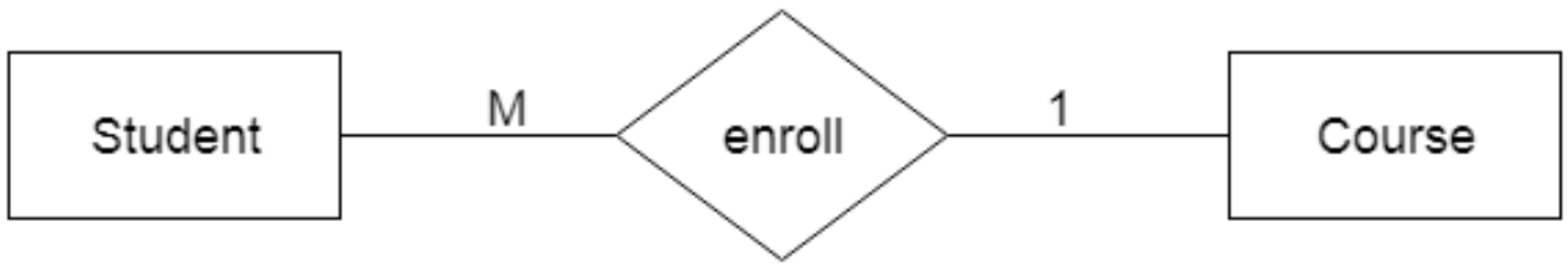
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

Summary

- Data model design is a 3 step process – conceptual, logical, physical (next time)
- Conceptual and logical data models can be expressed using Entity Relationship (ER) diagrams
- ER diagrams capture the entities, attributes, and relationships to model your information domain
- ER diagrams are a powerful way to document the design of your data model

Steps in Data Model Design

1. Identify entities
2. Identify relationships among entities
3. Determine the cardinality and participation of relationships
4. Designate keys / identifiers for entities
5. List attributes of entities
6. Identify constraints and business rules

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)

Thank you